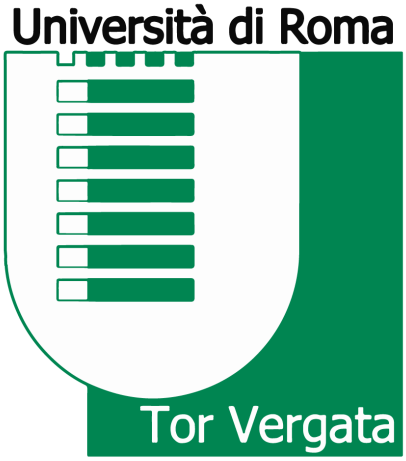

University of Rome Tor Vergata

Department of Mathematical, Physical, and Natural Sciences

An LLM-Based Approach for Explaining and Reducing the Clever Hans Effect

Submitted in Partial Fulfillment of the Requirements
for the

Bachelor's Degree in Computer Science

by

Luca Zhou

Supervisor: **Fabio Massimo Zanzotto**
Co-supervisor: **Michele Mastromattei (Ph.D.)**

July 2023

Abstract

Large pre-trained *deep neural networks* have achieved remarkable success in recent years, particularly in the field of *natural language processing* (*NLP*). However, alongside this progress, a phenomenon known as the *Clever Hans effect* has emerged. The term originates from a historical narrative in the 20th century. This effect occurs when a model performs exceptionally well on one dataset but fails to generalize to others due to its reliance on unintended cues learned from the former dataset, which are absent in the latter.

This work aims to address the *Clever Hans effect* in the context of a *Protein BERT* model pretrained on *TAPE* protein sequences. It begins by introducing *Explainable AI* and reviewing relevant prior research. Subsequently, it describes a data-refining pipeline to mitigate the *Clever Hans effect*. In essence, we unmask and remove the Clever Hans cues from a biased dataset by harnessing *Explainable AI* and a heavily biased model. A student model is then fine-tuned on the resulting cleaned dataset to imitate the biased model's predictions. The generalization performance of this student model is expected to considerably surpass that of its master, the biased model, and nearly as well as a healthy model trained on a cue-free dataset from the beginning.

Ultimately, the study further bolsters the potential of *Explainable AI* in unmasking and eradicating the Clever Hans effect from fallacious data.

Acknowledgement

Before beginning, I would like to express my sincere gratitude and heartfelt appreciation to my thesis supervisor Dr. **Fabio Massimo Zanzotto** and my co-supervisor **Michele Mastromattei** (Ph.D.). The topic of this thesis was proposed by Dr. Fabio Massimo Zanzotto who introduced me to the intersection between artificial intelligence and biology. Throughout this journey, I was guided by Michele Mastromattei who excelled at providing useful advice and encouragement, rendering this experience enjoyable and extremely educative.

I would also like to extend my thanks to the entire faculty of Science at the University of Rome Tor Vergata for creating a conducive academic environment and providing the necessary resources for the successful completion of my bachelor's studies. The surrounding community was stimulating and peer support from my dear classmates (especially Matteo Concutelli and Matteo Stromieri) has greatly enriched my learning experience.

Last but not least, I am indebted to my family for their unwavering support and comprehension throughout this challenging chapter of my life, as they have provided me with the time and resources to pursue my dream without putting pressure on me.

Contents

List of Figures

1. Introduction

The **Clever Hans effect** is a phenomenon of interest across various fields, including artificial intelligence. In the realm of AI, this effect has become increasingly prevalent, perhaps due to the vast amount of unanalyzed data fed into machine learning models. The term "Clever Hans" originated from a 1904 anecdote about a horse that seemingly possessed the ability to perform basic arithmetic operations. However, it was later discovered in 1907 that the horse had no understanding of numbers whatsoever; instead, it responded correctly thanks to cues from its owner's body language. This story serves as an analogy for machine learning, wherein a model achieves impressive performance by relying on unintended cues in the data that should not be relevant at all.

To illustrate this concept, consider training a model to distinguish between images of horses and cats. The model demonstrates exceptional accuracy on the provided dataset but fails drastically when presented with unseen data. Upon investigation, it is revealed that the model relied solely on the presence of a green background in horse images to make predictions. Consequently, the model did not actually learn any meaningful information about horses and cats, rendering it valueless in practice.

The motivation of this work is to investigate the Clever Hans effect in the field of *natural language processing* (*NLP*), specifically focusing on a *Transformer*-based model applied in the protein domain. Understanding this phenomenon is crucial for the broader acceptance of AI in critical fields like healthcare and business, where ensuring the model's reasoning is accurate is paramount. A mistakenly trained model can lead to disastrous outcomes, and exceptional performance on specific data should be met with skepticism rather than excitement.

Detecting the Clever Hans effect might be difficult in general, but some *Explainable AI* methods allow for beholding which parts of the input a model pays the most attention to, and in this work, two of them will be applied to confirm our intuition and one to remove the bias in the data.

The experimental task in this work is **binary classification**, where the model is tasked with predicting whether a given sequence of amino acids represents a real peptide. To investigate the Clever Hans effect, a deliberately designed artifact will be introduced, which the *Protein BERT* classification model heavily relies on after fine-tuning. The aim is to train a new model without this defect, on a refined dataset without the Clever Hans cue.

Several methods have been proposed to address the impact of the Clever Hans effect on models trained on affected data. However, most of these methods involve manipulating the training data and re-training, which may not always be feasible if the cue is unknown. In

contrast, this work describes a pipeline that focuses on a defective fine-tuned model and its training data and attempts to produce cleaner data and a better model out of them.

The outcome of this study suggests that biased data might still be valuable if properly transformed, and that *Explainable AI* is a powerful tool able to identify and eliminate the Clever Hans effect. The effectiveness of the pipeline proposed in this thesis hinges on the ability of the *Explainable AI* configuration to detect the model's reliance on cues in the data.

This thesis is structured as follows:

- **Background and Related Work**: a brief introduction to *Explainable AI*, some of its methods, and an attempt to address the Clever Hans effect;
- **Methods**: an overview of the model architecture utilized in this work, the source dataset, and the Explainable AI techniques exploited;
- **Experimental Procedure**: a detailed experimental description and the reasoning behind;
- **Experimental Results**: a report of the results attained from the experiments and observations.
- **Conclusion**: a recap and possible continuation work.

2. Background and Related Work

This section provides an introduction to the research area of *Explainable AI* and discusses some of its methods. Subsequently, it delves into a recently proposed method specifically designed to mitigate the Clever Hans effect.

2.1 Explainable AI

Current state-of-the-art machine learning algorithms have achieved remarkable success, often surpassing human-level accuracy and generating great excitement. However, when multiple algorithms perform similarly well on a given task, it becomes crucial to explore their properties beyond just *accuracy*. This includes considering additional performance metrics such as *precision*, *recall*, and *F1* score. However, one crucial aspect that is often overlooked is explainability. Explainability refers to the possibility for humans to understand the reasoning behind the decision-making process of machine learning models. The level of explainability varies greatly depending on the nature of the model.

We distinguish two types of machine learning models.

- A machine learning model is classified as **linear** if its output is specified as a linear combination of the input features.
- A machine learning model is instead classified as **non-linear** if its output cannot be computed as a linear combination of the input features. These models are relatively more complex but also more capable.

In general, linear models are naturally more explainable due to their simpler inner inference mechanism. The coefficients in the linear combination can be interpreted as weights of importance, where a larger coefficient for a feature indicates its greater influence on the output. On the other hand, non-linear models are often deployed as black boxes, as their reasoning mechanism is obscure for humans to comprehend. This lack of transparency presents a significant obstacle to the wider adoption of powerful non-linear algorithms in healthcare, as patients are hesitant to trust a program unable to explain its potentially hazardous decisions. As a comparison, *decision trees* are laudable interpretable and transparent models, as their decisions are based on the examples used during their construction. In contrast, *neural networks* are challenging to interpret due to their complex non-linear operations and optimization processes.

In recent years, there have been significant research efforts dedicated to making non-linear

models more interpretable, resulting in the development of powerful tools for exploring the inner workings of these black boxes. The following sub-sections will present several techniques for explaining machine learning decisions.

2.1.1 Permutation Importance

This technique is a simple method used to estimate the importance of each input feature. The importance of a feature is determined by its relevance to the final decision made by the model. The process involves randomly shuffling the values of a feature across the entire dataset and then measuring the decrease in the model's performance compared to a baseline. The underlying idea is that if a feature is influential, shuffling its values will significantly disrupt the model's performance, whereas an irrelevant feature will have minimal impact when corrupted. This procedure is repeated for each feature, allowing them to be ranked according to their importance.

2.1.2 Partial Dependence Plot

A partial dependence plot is a graphical representation that illustrates the relationship between one input feature and the model's output. The plot shows the output value as a function of a single feature while keeping all other features fixed. This technique helps us understand the relationship between each feature and the model's output. While it may not provide a complete explanation of the feature's effect on the output, it can estimate the direction and magnitude of its impact, whether positive or negative. When other features are held constant, their values are typically set as a design choice, such as their average values across the dataset. Partial dependence plots are useful for identifying important features, taking us a step closer to explainability.

2.1.3 Shapley Value

The notion of Shapley value originally stems from *cooperative game theory*. In the context of machine learning, it is used as a contribution attribution method for the features. Features act as players in the coalition of some sets of features, and the total payoff (prediction) is fairly distributed among them depending on their contribution. The concept is similar to permutation importance, we also want to rank the features according to their importance, which in this case is technically called the marginal contribution. If there are k features, we would have 2^k possible coalitions. Therefore, when the features are numerous, the computation can be expensive. Therefore, sampling techniques are usually used to approximate Shapley values by evaluating just some promising coalitions.

To calculate the Shapley value for a specific feature, we compute its marginal contribution across all coalitions in which it participates and then take the average contribution. The Shapley value of a feature tends to be higher if its presence has a significant impact, meaning that including or excluding the feature results in a noticeable difference in the model's prediction. Once each feature has its own Shapley value, we can rank them according to their contributions.

2.1.4 Layer-wise Relevance Propagation (LRP)

This model-specific method [5] is designed for explaining neural network predictions. It decomposes the output of a non-linear output in terms of the input features and produces a vector of feature scores. This vector explains how each input feature influences the final output.

Suppose the input consists of d features, and let x be an input vector of dimension d. Let *nn(x)* be the output of the neural network given input x. *LRP* decomposes *nn(x)* into d relevance scores, one for each input feature or dimension.

$$nn(x) = \sum_{k=1}^{d} R_k$$

That is, the output of the neural network is decomposed as a sum of d relevance scores, each representing the importance of one input feature.

The relevance scores of the input features cannot be directly obtained from the network's output. Instead, they are computed using a *backpropagation*-like mechanism. This process involves propagating the relevance scores backward from the output layer to the input layer. The relevance scores are propagated based on a conservation property, where each neuron in layer L propagates the relevance it receives from layer *L+1* to neurons in layer *L-1*.

Let j be a specific neuron in layer *L-1*, and let k denote arbitrary neurons in layer L, each neuron k propagates some of its relevance to neuron j

$$R_j = \sum_{k=1}^{|L|} \frac{z_{j_k}}{\sum_{i=1}^{|L-1|} z_{i_k}}$$

$|L|$ denotes the number of neurons in layer L and z_{j_k} represents the contribution of neuron j on making neuron k relevant. The denominator serves to enforce the conservation constraint. Hence, the relevance of neuron j is the scaled sum of contributions it makes to the neurons in the next layer, multiplied by the relevance scores of those next-layer neurons.

It can be proved that this mechanism does fulfill the conservation constraint, locally and globally.

Locally

$$\sum_{j=1}^{|L-1|} R_j = \sum_{k=1}^{|L|} R_k$$

Globally

$$\sum_{i=1}^{|input\ \ layer|} R_i = nn(x)$$

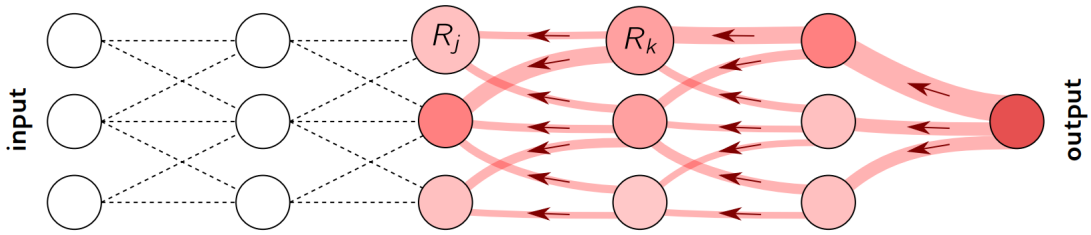

Figure 2.1: Backward Relevance Propagation

The general idea of *LRP* involves propagating relevance scores from the output layer to the input layer. However, different neural network architectures may require specific adjustments to the relevance backpropagation rule. In networks that use $ReLU$ non-linearities, there are three types of relevance propagation rules applied to individual layers based on certain conditions.

2.1.5 Local Interpretable Model-Agnostic Explanations (LIME)

This model-agnostic method [7] is based on a surrogate model and is very versatile. It can be applied to explain any supervised machine learning model and is particularly useful for explaining black-box predictions without requiring access to the internal workings of the model. The *LIME* (*Local Interpretable Model-Agnostic Explanations*) approach consists of training an interpretable surrogate model that explains the behavior of the black box model in the vicinity of a specific prediction instance. The interpretable model can be of any type, as long as it is easily understandable by humans.

To explain the local behavior of the black box model around a particular instance, the training dataset for the interpretable model is created by perturbing the input points near the instance. The outputs of the black box model on these perturbed points are also included in the training data. Various local perturbations can be applied, such as removing parts of the input or slightly modifying the values of certain input features. Each perturbed test point is likely to yield a different prediction from the black box model. These predictions are then weighted based on the proximity of the perturbed points to the original instance. Test points that are closer to the original point have higher weights in the calculation of loss.

Let pink and blue background colors represent the model's binary prediction in a 2D plane, and let the red cross X be the prediction point we want to explain. The test points around X are sized according to their weight, that is, their proximity to X. *LIME* in this example builds an interpretable linear model to estimate just the local predictions around X. This linear model explains the local behavior of the model, even though it fails outside the neighborhood.

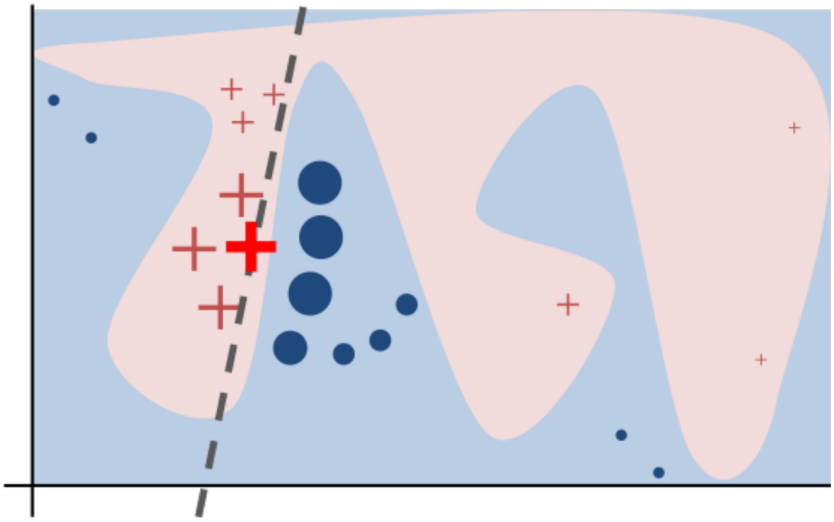

Figure 2.2: Local Interpretable Linear Model

2.1.6 Self-Attention

The concept of *self-attention* is introduced with the *Transformer* [8] architecture, which will be investigated in depth in the next section. The goal of this sub-section is to introduce the basic idea behind *transformer*-based models. Initially designed for language-related tasks, such as *language understanding* and *question answering*, *transformer* models aim to comprehend natural language sentences holistically and capture the relationships between individual words and their meanings.

One key feature of transformer models is the *self-attention* mechanism. This neural network module allows each input word to be compared against all other words, including itself, in the input sentence. It then calculates a weighted "sum" across all words based on their contextual relevance to the current word. This enables the model to capture the meaning of word combinations and understand idiomatic expressions, such as the figurative meaning of "*break a leg*" as "*good luck*".

To summarize, the *self-attention* module takes an input sequence of words without contextual information and generates an output sequence of words where each word does have contextual information. In practice, the input elements are represented as word *embedding vectors*, and the weighted sum refers to the aggregation of these vectors based on their respective weights.

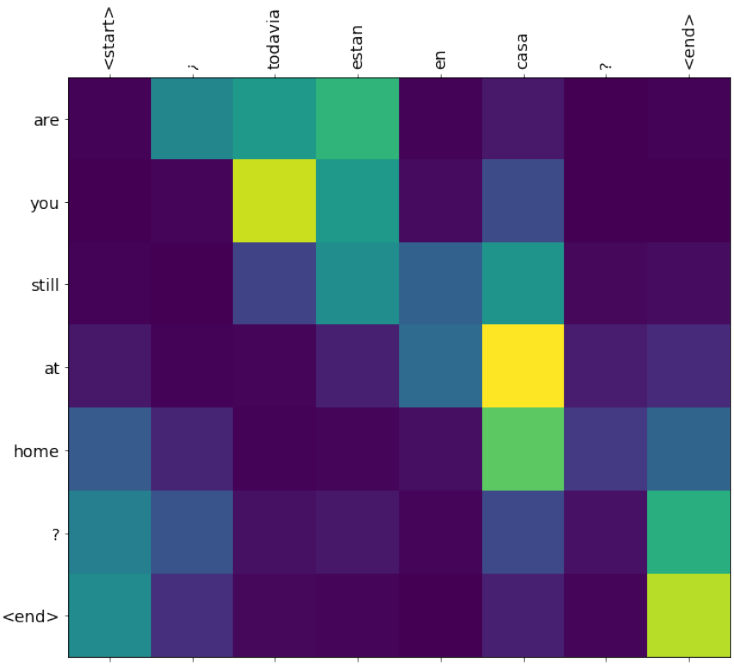

Figure 2.3: Self-attention visualization in Neural Machine Translation

2.1.7 Occlusion Sensitivity

This explanation technique is proposed in [9] and is commonly applied to *computer vision* tasks to understand the attention of an image classification model. This method slides a grey square across the input image, sequentially occluding different portions of the image, and observes how the model's prediction alters. The underlying idea is straightforward: if an important region of the image is occluded, we would expect a noticeable decrease in the quality of the model's prediction.

The output of this process is a *heatmap*, which is a visual representation of the input image. In the heatmap, each cell corresponds to a specific location in the image, and the value in each cell represents the predicted probability of the correct class. By analyzing the *heatmap*, we can identify the regions of the image that the model attends to the most when making its prediction.

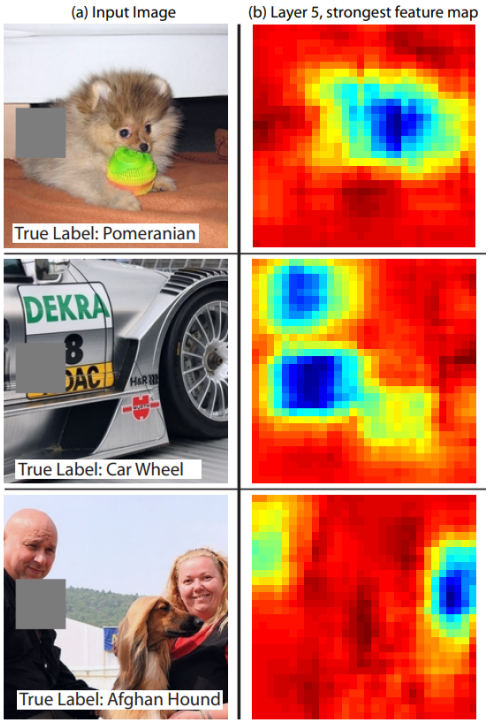

Figure 2.4: Occlusion Sensitivity

2.2 Clever Hans Effect Mitigation

Clever Hans effect and *XAI* are tightly related as the latter can serve to unmask the former, but once unmasked, removing or at least alleviating the problem is an even harder task since it also intrinsically relates to biases in the training data.

Christopher J. Anders et al. define, in their paper [1], a Clever Hans artifact as a pattern in the training data that suggests to the model the correct prediction. Once an artifact has been detected, the goal becomes making the model more or completely insensitive to them.

Let h_{art} be a model that takes in input a non-artifact example and returns an artifact example, that is, it turns a correct example into one affected by C.H. effect. Its learning objective is defined as minimizing the L2 distance between real artifact examples and produced examples. Let X^+ be the set of examples subject to the C.H. artifact and X^- of those without.

$$\hat{\theta}_{art} = argmin_\theta \frac{1}{|X^-||X^+|} \sum_{x^- \in X^-} \sum_{x^+ \in X^+} ||h_{art}(x^-; \theta) - x^+||^2$$

Therefore, the sensitivity of a function or model f to a particular C.H. effect modeled with h_{art} is estimated by

$$S_{art} = \frac{1}{|X^-|} \sum_{x^- \in X^-} ||f(h_{art}(x^-; \hat{\theta}_{art})) - f(x^+)||$$

The intuition behind adding a C.H. effect instead of removing it is that the introduction of an artifact is generally more feasible than its removal.

The same paper then provides some key insights into the desensitization of the C.H. effect depending on the scenario. A learning model's behavior can be rectified by modifying its training data. Here they posited the following approaches:

- If data is enough, then naively exclude all examples affected by the C.H. effect.
- If affected examples outnumber non-affected ones, discard all non-affected ones so that the learned model becomes artifact-insensitive.
- If the presence of the C.H. effect is non-invertible, turn all examples into C.H.-affected.
- If the presence of the C.H. effect is invertible, turn all CH-affected examples into non-affected ones.

In the case of classification tasks, if we decide to artificially introduce artifact examples, then the same number of those should be added to each class for the sake of balance.

The paper proposed the *Augmentative Class Artifact Compensation* (*A-ClArC*), a systematic method for modifying the training data to reduce or remove the C.H. effect. The goal is to render an SGD-trained classifier less sensitive to the CH artifact. Abstractly, we can think of a C.H. artifact as having a direction that deflects the decision of the model.

Let our problem be binary classification and let A and B be the two classes. Assume A is the class affected by a CH artifact c. *A-ClArC* is an algorithm that injects an artificial artifact into class-B examples that has the same artifact direction as c. The outcome of this procedure significantly alleviates the impact of c on the model.

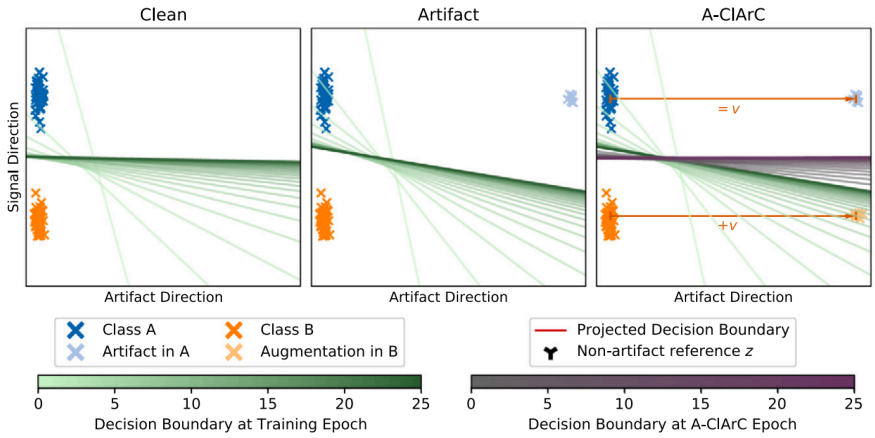

Figure 2.5: A visual rectification effect of A-ClArC

This work corroborated the importance of training data in the Clever Hans effect and that such undesired phenomena can be effectively mitigated by modifying the training data in a clever way.

3. Methods

In this section, the ingredients of the experimentation are illustrated. They encompass the models used, the methods employed, and the source dataset.

3.1 Transformer

Before the advent of *transformers* [8], state-of-the-art sequence modeling architectures mainly consisted of *recurrent neural networks*, *long-short term memory* [4], and *gated networks*. What they all have in common is the sequential nature, that is, hidden states are elaborated in sequence. The hidden state at step t depends on the hidden state at its previous step *t-1*. Drawbacks of this approach include the non-parallelizability of computation, inability to maintain a long memory, and *exploding/vanishing* gradients in *backpropagation*. The advent of the *transformer* has been revolutionary, especially in language-related tasks. It brought a significant innovation by relying entirely on the *attention* mechanism and eliminating recurrence. This departure from sequential processing allowed for more parallelizable computations, enabled the model to capture long-range dependencies effectively, and mitigated the issues of gradient *vanishing* and *exploding*. The transformer architecture has shown remarkable performance and has become the go-to choice for many tasks even beyond the realm of natural language processing.

3.1.1 Architecture

The architecture of the transformer is composed of N encoders followed by N decoders and custom layers depending on the task. For instance, if the task is text classification, on top of the last *transformer* decoder, a *dropout* layer can be applied to prevent overfitting, a fully-connected layer can be used to map the decoder output to the desired output dimension, and a *sigmoid* activation function can be applied to produce the classification probabilities.

Each encoder receives a continuous representation in input, elaborates it, and passes it to the next encoder or to the decoders if the encoder is the last. A decoder behaves similarly but generates an output sequence of elements one by one autoregressively. Autoregressive means that its previously generated output is also an additional input when generating the current one.

Inside an **encoder** there are two sequential sub-layers:

- Multi-head self-attention

- Fully-connected feed-forward network

Each sub-layer is followed by **layer normalization** operation supported by a *residual* connection, where we normalize the sum of the output and input of that sub-layer.

A **decoder**, however, has three sequential sub-layers:

- Masked multi-head self-attention
- Encoder-decoder multi-head attention
- Fully-connected feed-forward network

The first decoder sub-layer is indeed similar to the one in an encoder. However, there is a key difference in that the decoder self-attention mechanism has a *masked attention mask* that prevents the decoder from attending to future positions. This ensures that the decoder can only rely on the information available up to the current position and cannot "cheat" by accessing future outputs during generation. The second sub-layer in a decoder is the *multi-head attention* mechanism, which performs attention over the output of the encoders. This allows the decoder to focus on different parts of the input sequence during the decoding process. The final *fully-connected feed-forward* network then generates the output. Again, the authors of the paper deploy a *layer normalization* operation after each decoder sub-layer.

Before the input is processed by the *transformer*, it undergoes two steps: *word embedding* and *positional encoding*. Word embedding converts the raw input into a sequence of **embeddings**, representing the meaning of each word. *Positional encoding* adds information about the positions of words in the sequence. The resulting input is a position-aware sequence of embeddings, which is then ready to be processed by the transformer.

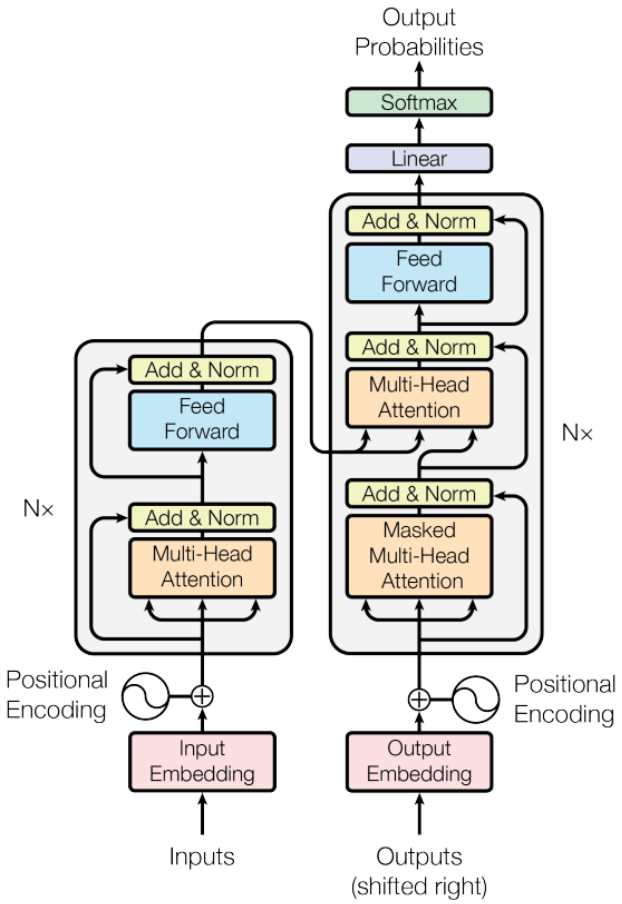

Figure 3.1: Transformer architecture

3.1.2 Attention Mechanism

Thus far, we have interpreted the attention mechanism as a way of attributing importance to all input words when processing one word. In practice, instead of relying on a single attention mechanism, the input is processed through multiple attention heads, each with its own set of parameters. There comes the name *multi-head*. It allows the model to capture different types of dependencies and relationships within the input sequence, enhancing its ability to learn complex patterns and improve performance.

The attention is calculated via the following three vectors for each input element:

- *Query*
- *Key*
- *Value*

In short, let x be the word being currently processed, its query vector is calculated by multiplying x by a learnable query matrix Q. Likewise, its *key* and *value* vectors are computed by multiplying x by their respective learnable matrices K and V. The result of applying self-attention on x is a weighted sum of the *value* vectors of all words (including itself), where

the weights are calculated by multiplying the query vectors by the key vectors. So, given an element x in the input, x is transformed into a new vector that incorporates the context from all input elements according to their relevance.

One major advantage of the transformer architecture is the potential for parallel computation. The *self-attention* sub-layer, in particular, allows for high parallelizability as each embedding vector in the input can be processed independently, after computing the three vectors for each input embedding. Similarly, the *fully-connected* sub-layer can handle one embedding vector at a time. However, it's important to note that the transformer architecture is not fully parallelizable since the input still needs to sequentially flow through the encoders and decoders.

3.2 BERT

Macroscopically, the transformer can be divided into the encoder stack and decoder stack. The *Bidirectional Encoder Representations from Transformers* (*BERT*) [2] is a language model based just on the transformer encoders. As with any language model, it gains some understanding of human natural language and can be pretrained and then efficiently adapted to downstream tasks such as language translation and sentiment analysis.

The unsupervised pretraining objectives of *BERT* are

- Masked language modeling (*MLM*)
- Next sentence prediction (*NSP*)

In *MLM*, some percentage of input tokens are masked or permuted, and the model is trained to predict those masked tokens. The model is expected to gain some understanding of the language. In *NSP*, the model instead learns sentence relationships. Here, two sentences are provided as input and the model predicts whether the second sentence logically follows the first one.

To adapt a *BERT* model to downstream tasks, it is fine-tuned on a task-specific dataset. A language model that already possesses language understanding requires just a little tweak to reach high performance. There are two common fine-tuning approaches. The first simply adds a task-specific layer on top of a pretrained *BERT* model and only allows the model to learn the weights for that layer while freezing the pretrained layers. The second also adds that specific layer but allows the whole model, including the pretrained base *BERT* layers, to be optimized. The second approach usually yields better results.

Allegedly, earlier layers learn more basic features in the language such as grammar, middle layers learn syntax, and deeper layers learn abstract semantic features. Thus, we would expect that fine-tuning a pretrained *BERT* model on a downstream dataset would cause larger parameter updates in the deeper layers because the grammar and syntax normally do not differ significantly from task to task.

3.3 TAPE Protein BERT

In this work, I will focus the experiments on protein sequences consisting of amino acids. This bears some resemblance to natural language processing since protein sequences can be viewed as sentences where the amino acid symbols play the role of words. *TAPE (Task Assessing Protein Embeddings)* [6] is a framework that introduces five relevant benchmarking tasks in the biological domain to facilitate the comparison between different protein models. More importantly for us, it enables us to adapt BERT to learn protein representations. The model used in the following experiments is Protein BERT, a variant of BERT based on the TAPE tokenizer and pretrained on a large unlabeled protein dataset.

3.4 Source Dataset

In this work, the peptide identification datasets are derived from the "*Anticancer Peptides*" dataset provided by the *Center of Machine Learning and Intelligent Systems* at the University of California Irvine. The original dataset consists of approximately 1000 peptide sequences, each with an associated *ID* and a *class* indicating peptide activity. For our experimental datasets, only the *peptide sequence* field was extracted from the source dataset. This subset represents the portion of our datasets where the sequences are labeled as true peptides, since they are authentic sequences of amino acids rather than arbitrary sequences.

3.5 Explanation Techniques

In the following experiments, some results and intuition will be corroborated by visualizing **self-attention** and a variant of **permutation importance**. Self-attention values will be plotted in a heatmap, where the entry *(i, j)* represents the amount of attention given to symbol j when processing symbol i. The latter technique consists of randomly permuting the value of an input symbol and recording the model's drop in prediction confidence with respect to the correct class. The most important symbols will have caused the most severe drop in model confidence. This is regarded as a variant of permutation importance because it does not permute the values along each feature, but the value of each input token instead. For simplicity, we will still refer to this method as permutation importance.

4. Experimental Procedure

This section illustrates the whole experimental process I have conducted to attain the final results and speculations. Overall, I will first create a dataset for peptide identification, where true peptides are labeled as 1 and fake ones as 0. The true peptide examples in this dataset are extracted from the source dataset from UCI, whereas the fake examples are generated as random sequences of *IUPAC* amino acid symbols. I will then create a second dataset similar to this one but with a slight modification. Only for examples of true peptides, I will replace their first three symbols with "CHE" (Clever Hans effect), an arbitrary choice. This is a Clever Hans artifact that suggests the correct prediction. Now, these are two peptide identification datasets with over 1900 examples, half of which are true, and each will be used for fine-tuning one Protein BERT model. We will refer to the model trained on the "CHE" Hans dataset as the Hans model and the other as the healthy model. Permutation importance on the Hans model will be used to unmask and remove the C.H. artifact from the Hans training set, resulting in a "cleaned" version that hopefully resembles the normal dataset. This "cleaned" dataset is then used to fine-tune a third model (called the student model as it imitates the decisions of the Hans model). The performance of the healthy model serves as an upper bound for the student model.

Overall, the steps taken in this process can be summarized in the following way.

- Creating two datasets
- Creating a custom model architecture
- Fine-tuning Hans and healthy models
- Corroborating the Clever Hans Effect
- Cleaning the Hans training set
- Fine-tuning the student model

Next, each step will be elucidated in a dedicated sub-section.

4.1 Creating Datasets

As the first step, the source peptide dataset [3] is downloaded from the website of the "*Center of Machine Learning and Intelligent Systems*" at the University of California Irvine. This dataset contains nearly 1000 peptide sequences with their respective *ID* and peptide *activeness*.

	ID	sequence	class
0	1	AAWKWAWAKKWAKAKKWAKAA	mod. active
1	2	AIGKFLHSAKKFGKAFVGEIMNS	mod. active
2	3	AWKKWAKAWKWAKAKWWAKAA	mod. active
3	4	ESFSDWWKLLAE	mod. active
4	5	ETFADWWKLLAE	mod. active

Figure 4.1: Source Dataset from UCI

Since *ID* and class do not matter for our purposes, I have dropped them from the dataset and only kept the peptide sequences that we know are authentic.

Next, I labeled each true peptide example as 1 and created random fake amino acid sequences with the label 0. The number of fake examples is the same as that of true ones, and the size of the union dataset is thus 1938.

	sequence	is_peptide
0	IQHKKHYGTRLYNWIWWMTCCGVT	0
1	ETNIRVALEKSFL	1
2	YSDPSQAYPATLCHRAPDKNHYG	0
3	AFRHSVKEELNYIRRRLERFPNRL	1
4	PEENFNAEVINQHTRNWQHPN	0

Figure 4.2: Normal dataset for peptide identification

The second dataset I have created is the Hans dataset. It is simply the same dataset as the previous one where each true example has its first three symbols replaced by "CHE".

	sequence	is_peptide
0	IQHKKHYGTRLYNWIWWMTCCGVT	0
1	CHEETNIRVALEKSFL	1
2	YSDPSQAYPATLCHRAPDKNHYG	0
3	CHEAFRHSVKEELNYIRRRLERFPNRL	1
4	PEENFNAEVINQHTRNWQHPN	0

Figure 4.3: Hans dataset for peptide identification

4.2 Creating a Custom Model Architecture

To adapt a pretrained "*bert-base*" Protein BERT model to our peptide identification task, a custom model is defined to enclose a base Protein BERT model with an additional *dropout* layer followed by a task-specific classification *linear layer*.

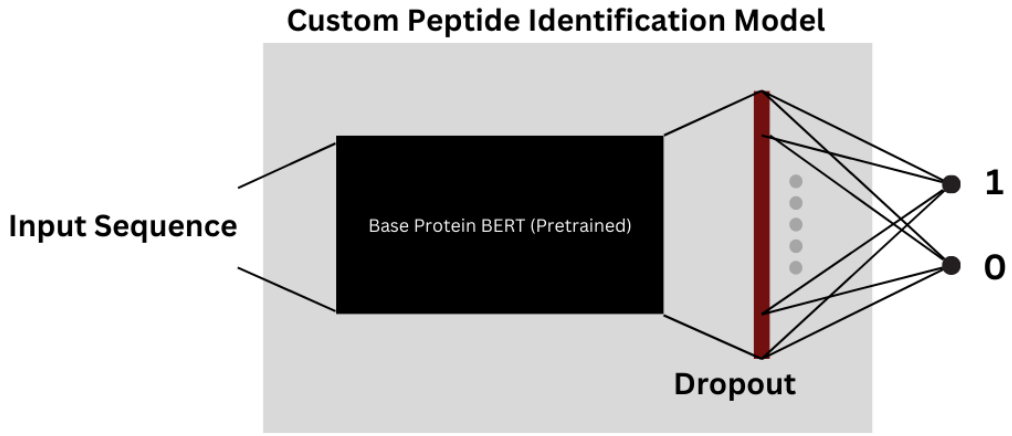

Figure 4.4: Custom model architecture for peptide identification

The model configuration is listed below.

- Base model: "*bert-base*" Protein BERT
  - 12 layers in the encoder stack
  - 768 hidden units in the feed-forward networks
- Dropout rate: 0.2
- Optimizer: *AdamW* (*Adam* with weight decay)
- Learning rate: 10^{-8}
- Tokenizer: TAPE Tokenizer with IUPAC vocabulary
- Input sequence padding: to 42

4.3 Fine-tuning Hans and Normal Models

We fine-tune two instances of the above-defined custom model, each on one of the two datasets. The specific configuration applies to both models and is reported below.

- Validation ratio: 0.2
- Batch size: 16
- Epochs: 5
- Loss function: *Cross Entropy*

After 5 epochs of fine-tuning, we show the performances of the two models on their respective validation data. For the Hans model, the performance will be reported both on Hans and non-Hans validation data.

Class	Precision	Recall	F1	Accuracy	Loss
0	0.948696	0.925652	0.933913		
1	0.923478	0.949130	0.931739		
				0.942174	0.0499

Table 4.1: Normal validation data performance: healthy model

Class	Precision	Recall	F1	Accuracy	Loss
0	1.0	1.0	1.0		
1	1.0	1.0	1.0		
				1.0	0.0002

Table 4.2: Hans validation data performance: Hans model

Class	Precision	Recall	F1	Accuracy	Loss
0	0.536522	1	0.691739		
1	0.086957	0.009565	0.017391		
				0.5399	4.3799

Table 4.3: Normal validation data performance: Hans model

As expected, the healthy model performed reasonably well on regular validation data achieving over 94% accuracy, whereas the Hans model performed flawlessly on the Hans validation data but poorly on normal data (below 0.54 accuracy). This suggests that the Hans model failed to properly learn the underlying amino acid pattern that makes up a peptide. Instead, it blindly and almost entirely relied on the Clever Hans artifact of "CHE". This obviously renders the Hans model undeployable and valueless in practice.

4.4 Corroborating the Clever Hans Effect

To confirm the speculation that the Hans model relies on the Clever Hans artifact "CHE", this section-section is devoted to comparing the Hans model against the healthy model via self-attention activations and a variant of permutation importance.

First, to prove that the Hans model has a different focus than the healthy model, we inspect the self-attention matrices in the last layer in their encoder stack. To do so, we compare the **heatmaps** of those matrices between the two models, on the same input.

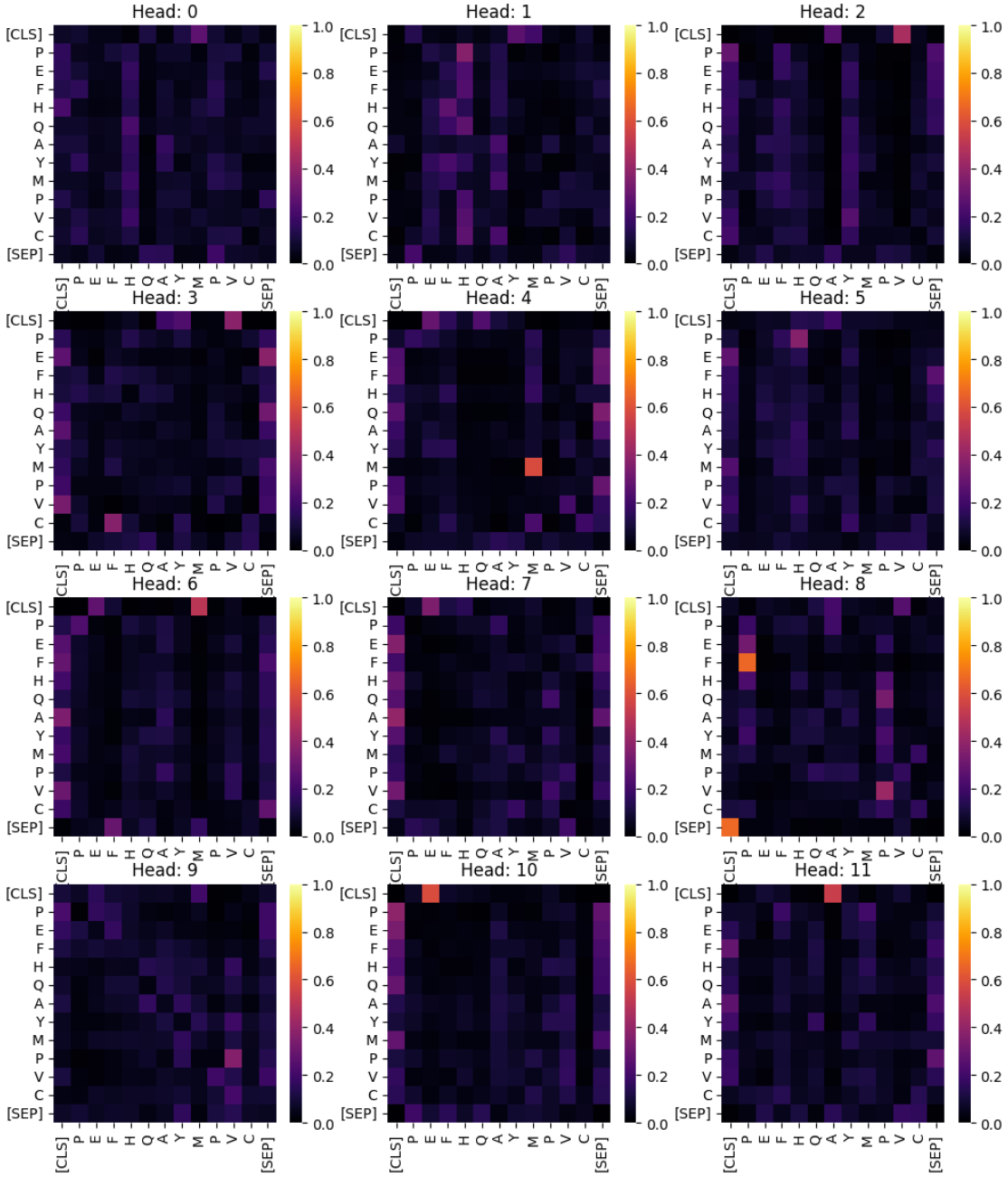

Figure 4.5: *PEFHQAYMPVC* (true peptide), Healthy model

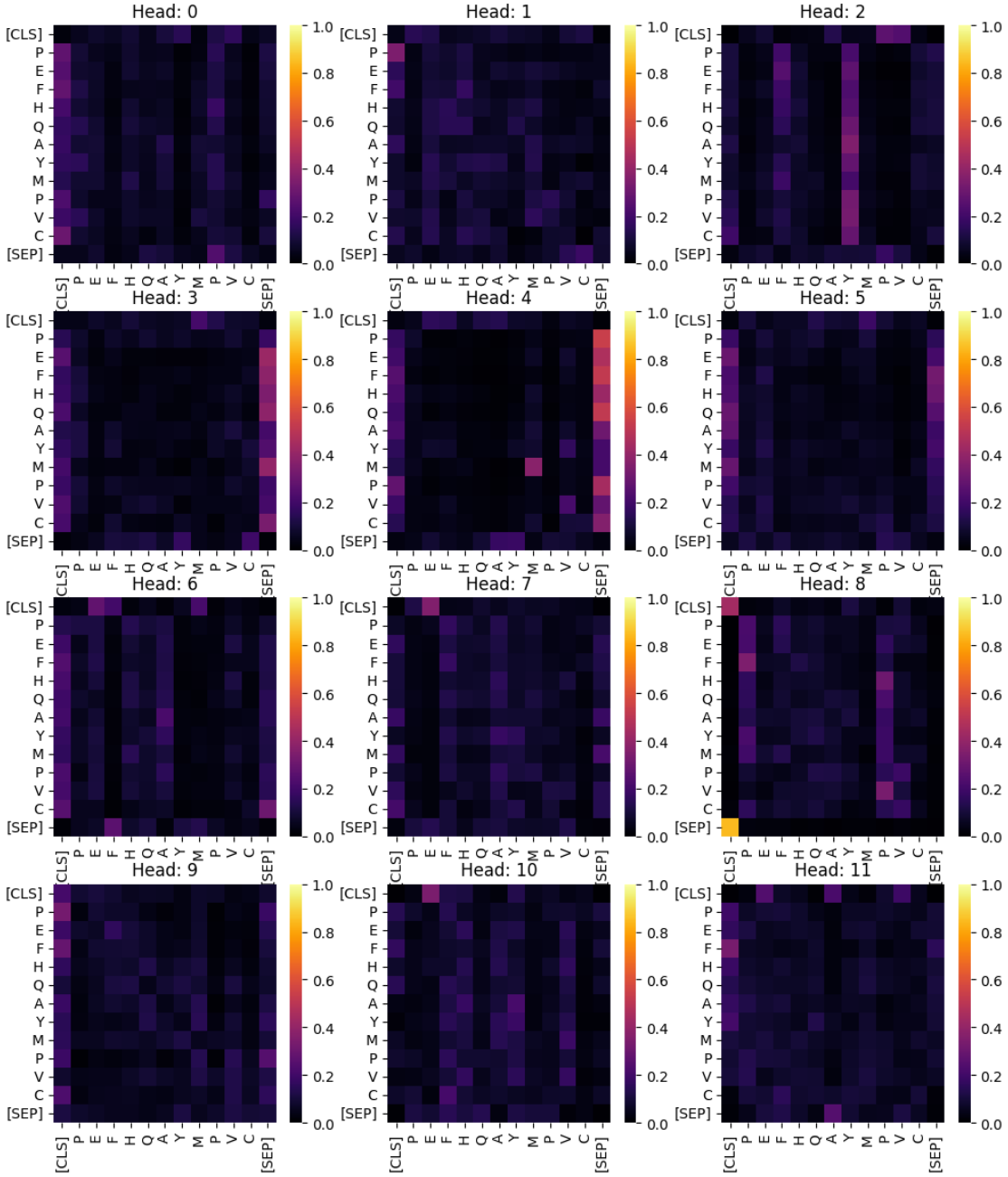

Figure 4.6: *PEFHQAYMPVC* (true peptide), Hans model

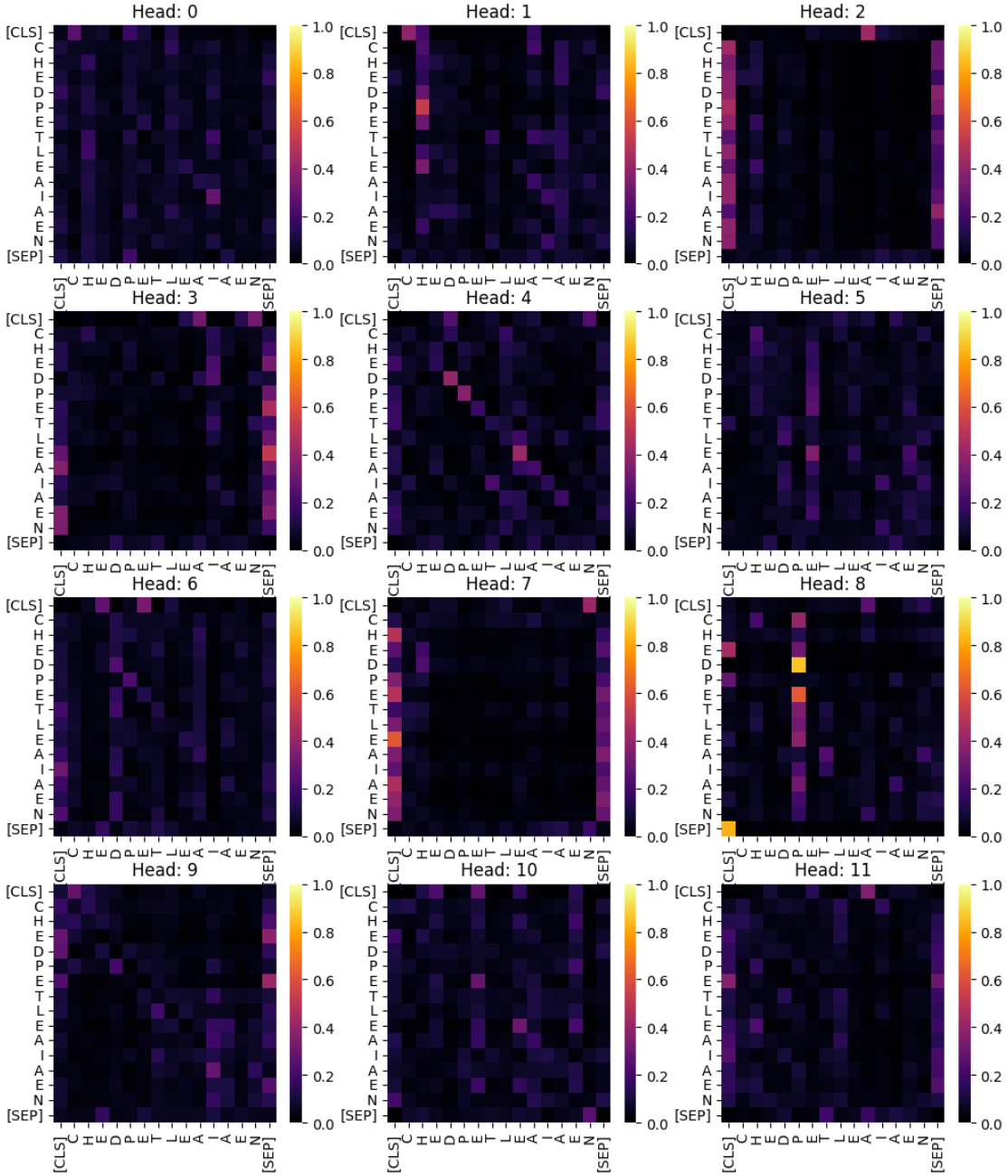

Figure 4.7: *CHEDPETLEAIAEN* (true peptide but with artifact), Healthy model

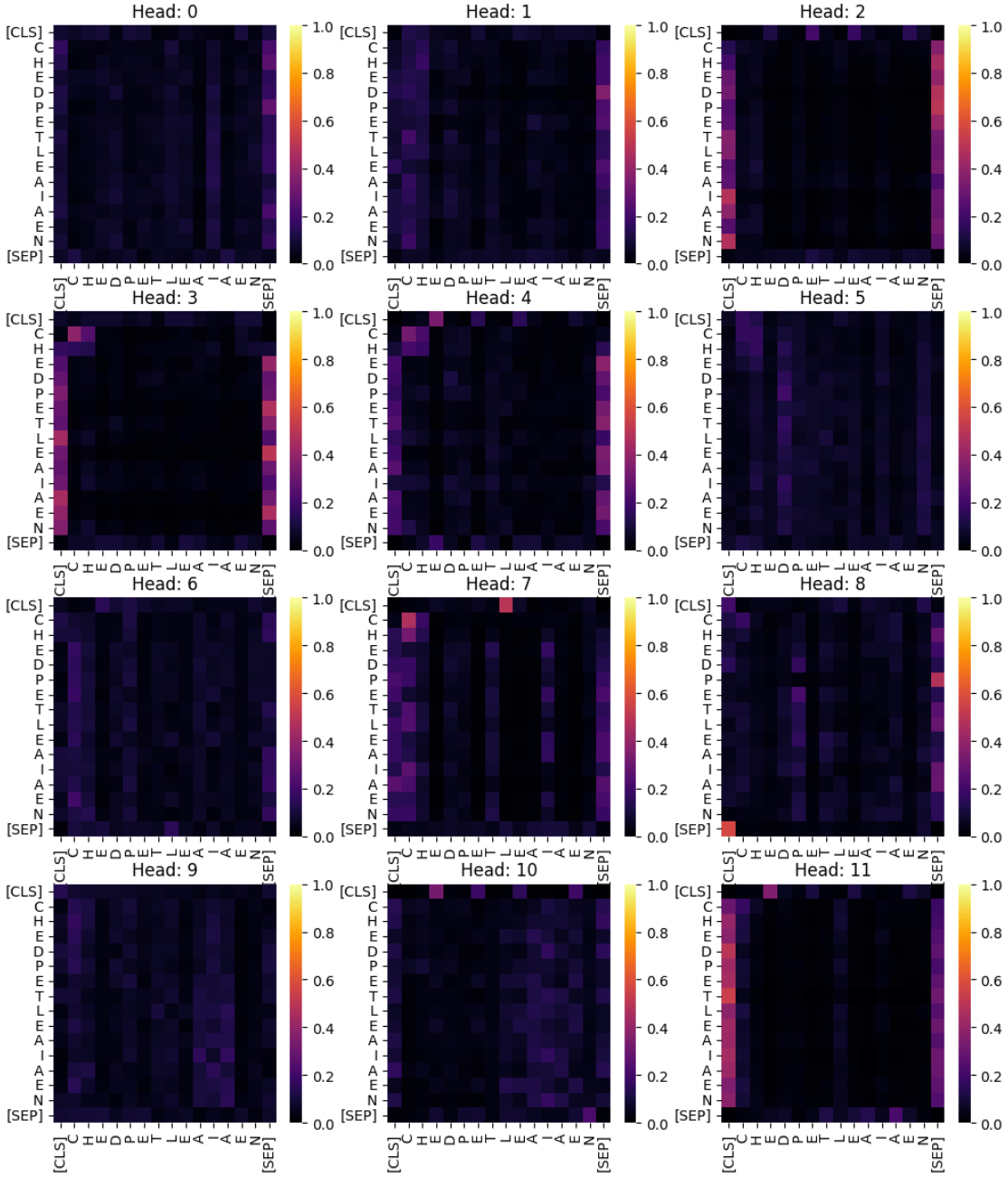

Figure 4.8: *CHEDPETLEAIAEN* (true peptide but with artifact), Hans model

Although the above heatmaps cannot explain the exact focus of the two models, we can behold that at the last layer, the healthy model tends to have more widespread self-attentions over the input, whereas the Hans model's attention is narrower, with fewer noticeable self-attention values in the heatmaps.

Next, we analyze the importance of each input symbol using **permutation importance**.

Given a sequence, we permute each of its symbols and record the drop in prediction confidence of the correct class. Note that a negative drop indicates a gain in confidence. The larger the drop caused by a symbol's permutation, the more relevant that symbol is for the model to predict correctly. The permutation of a symbol consists of replacing that symbol with each symbol in the *IUPAC* vocabulary, including the empty symbol, and averaging the confidence alteration.

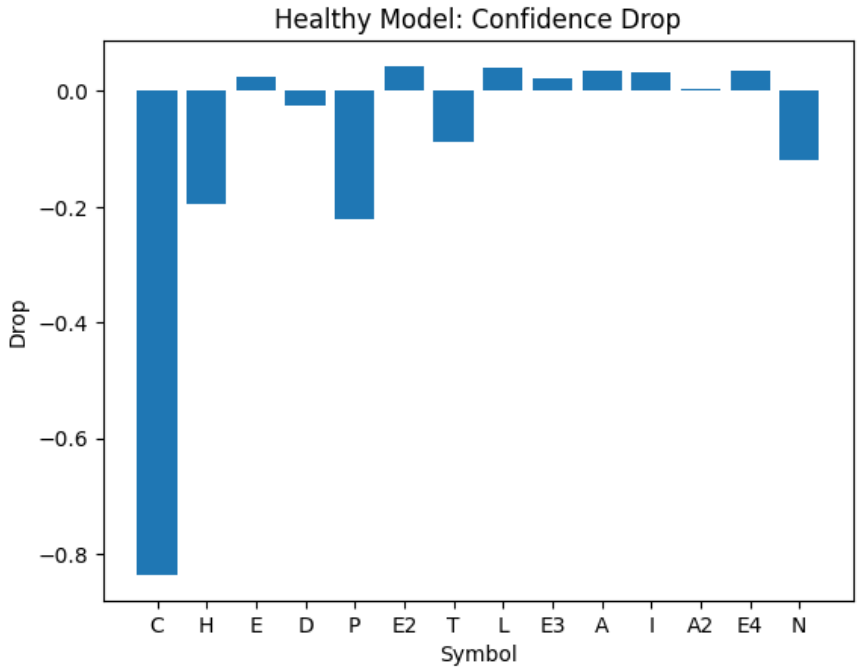

Figure 4.9: *CHEDPETLEAIAEN* (true peptide with C.H. artifact), Healthy model

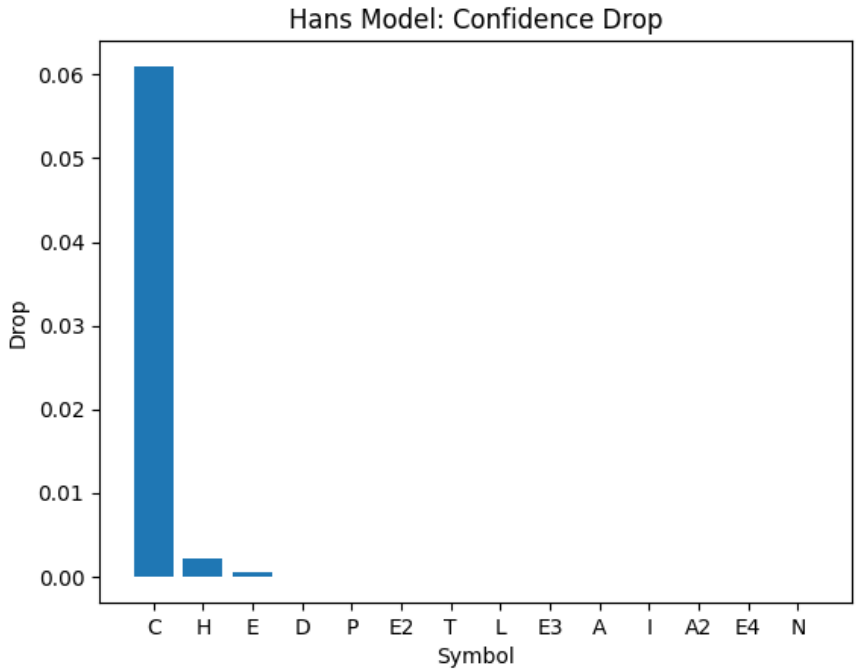

Figure 4.10: *CHEDPETLEAIAEN* (true peptide with C.H. artifact), Hans model

By comparing the two bar charts, it is clear that the Hans model pays much more attention to "CHE" in order to predict correctly, whereas the healthy model's confidence is irregular due to the fact that the input, as it is given, is actually not a true peptide.

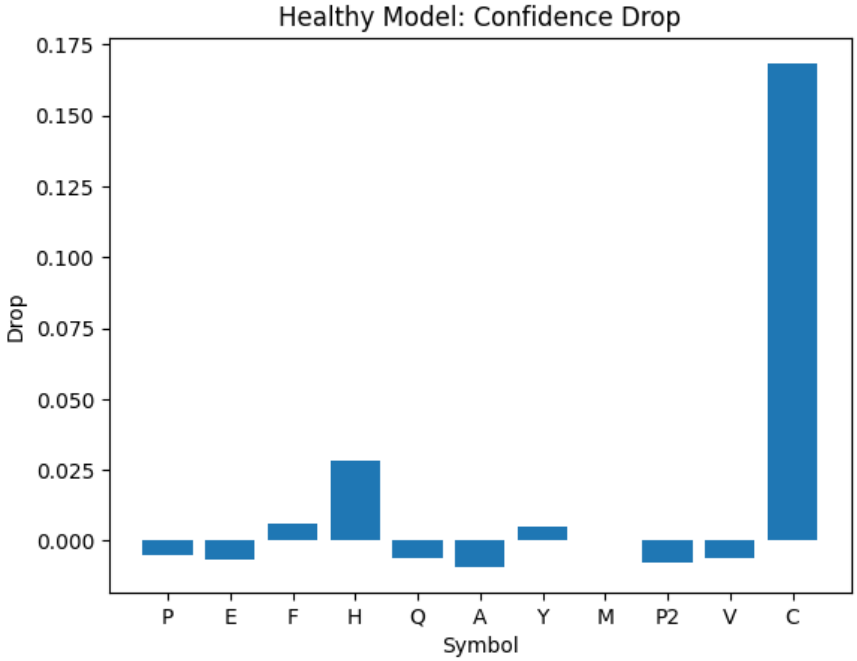

Figure 4.11: *PEFHQAYMPVC* (fake peptide), Healthy model

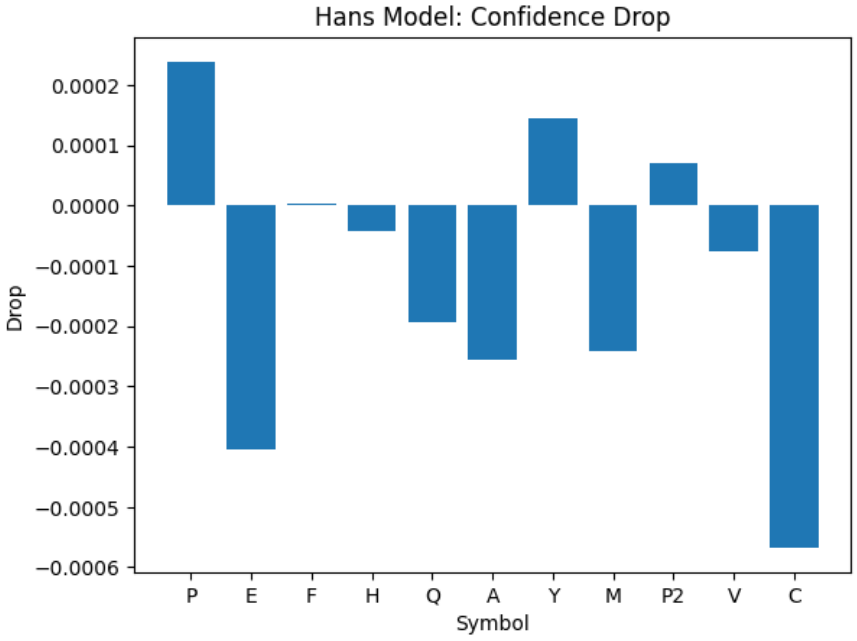

Figure 4.12: *PEFHQAYMPVC* (fake peptide), Hans model

In this last example, the permutation importances seem somewhat arbitrary. This agrees with the fact that the Hans model generally yields unpredictable or arbitrary confidence

when the artifact of "CHE" is absent.

4.5 Cleaning the Hans Training Set

To obtain a cleaned version of the Hans training set that resembles the normal training set, we utilize the Hans model to predict again all the examples in its training set, examples that it has seen many times already during fine-tuning. For each example, we use **permutation importance** to detect a subsequence of the input that Hans model heavily attends to. By removing that subsequence, we produce a "cleaned" example each time and add it to a new dataset. After iterating through the entire Hans training set, we obtain a cleaned training set that should resemble the normal training set without "CHE".

There are two **hyperparameters** that require domain knowledge and scrupulous selection:

- an **importance threshold** that determines when a subsequence is regarded as important enough to be a C.H. artifact;
- the C.H. artifact's **length**.

The two hyperparameters are intertwined. Assume that the length of the artifact is *3*, as is in our experiments, and the importance threshold is *0.95*, then the artifact removal of one example works as follows:

- use Hans model to classify the peptide;
- use permutation importance to compute the importance of all tokens;
- pick the top-three ranking tokens;
- remove the top-three tokens from the peptide if their cumulative importance exceeds *95%* of the total importance, otherwise leave the peptide as it is;
- the resulting example is either artifact-free already or unbiased and added to a new training set.

In general, when the length of the artifact is unknown or variable, it is possible to pick a set of plausible lengths, each with its associated importance threshold. This results in a set of potential artifacts, and a further algorithm is required to select the most likely one. This adaptation, however, weighs on efficiency.

To evaluate the generalization performance of the Hans model after manipulation, we run it on the normal validation dataset, where the C.H. artifact is absent.

The product of this pipeline is a new artifact-free training set on which we train the student model. The performance of this model hinges on the choice of hyperparameters in the pipeline.

	sequence	is_peptide
0	IQHKKHYGTRLYNWIWWMTCCGVT	0
1	ETNIRVALEKSFL	1
2	YSDPSQAYPATLCHRAPDKNHYG	0
3	CHEAFRHSVKEELNYIRRRLERFPNRL	1
4	EENFNAEVINQHTRNQHN	0
5	CHEFLGALFKALSHLL	1
6	DPETLEAIAEN	1
7	SLEQQFSIRSFATQVQNM	1
8	GTVCYTEPRFFKIHRSE	0
9	SFCPRFDKNHDIPHID	0

Figure 4.13: Unbiased training set

Note that not always does the removal procedure succeed in removing the "CHE" artifact.

4.6 Fine-tuning the Student Model

The student model is trained on this new unbiased training set and evaluated on normal data without "CHE". It is initialized in the exact same way as the previous two models and the fine-tuning setting is also the same (batch size = 16, epochs = 5).

5. Experimental Results

The validation result of the student model is reported below.

Class	Precision	Recall	F1	Accuracy	Loss
0	0.952217	0.916957	0.928696		
1	0.922174	0.966087	0.940870		
				0.942174	0.2649

Table 5.1: Normal validation data performance: Student model

For comparison purposes, the performance of the healthy model is again shown.

Class	Precision	Recall	F1	Accuracy	Loss
0	0.948696	0.925652	0.933913		
1	0.923478	0.949130	0.931739		
				0.942174	0.0499

Table 5.2: Normal validation data performance: healthy model

5.1 Observations

Based on the above results, we can observe that:

- the accuracy performance of the student model matched the healthy model's, although it is generally expected to fall a bit below;
- the loss of the student model, however, did not get as close as the accuracy did, indicating a higher degree of perplexity when predicting correctly;
- one possible speculation for why the student model performs this well is that it has been trained more robustly on a dataset with some noise, since the artifact has sometimes failed to be removed from the examples.

6. Conclusion

This work has first explored the area of **Explainable AI** and some methods therein, as well as the undesired phenomenon of the **Clever Hans effect**. In contrast to the more common data manipulation and re-training approach, we discussed a pipeline that works on top of the fallacious data and an ill-trained model. The experiments showed that the biased dependence of a model can be exploited to unmask and remove the underlying C.H. artifact from the data, leading to the creation of a new artifact-free dataset.

Since the experiments were not exhaustive, three interesting continuation works might be

- investigating how would other *Explainable AI* methods unmask the C.H. artifact;
- exploring different criteria for searching and classifying a subsequence as a C.H. artifact;
- developing a systematic way to choose the optimal hyperparameters (*importance threshold* & *artifact length*).

Finally, this work also encourages the discovery of procedures for mitigating incorrect model behaviors without wasting the fallacious data on which the model is trained. Instead, refining what is available might be a promising way out.

Bibliography

[1] Christopher J. Anders, Leander Weber, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. Finding and removing clever hans: Using explanation methods to debug and improve deep models. *Information Fusion*, 77:261–295, 2022.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Francesca Grisoni, Christopher S Neuhaus, Mitsugu Hishinuma, Gisela Gabernet, Jan A Hiss, Masaaki Kotera, and Gisbert Schneider. De novo design of anticancer peptides by ensemble artificial neural networks. *Journal of Molecular Modeling*, 25(5):112, 2019.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[5] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. *Layer-Wise Relevance Propagation: An Overview*, pages 193–209. Springer International Publishing, Cham, 2019.

[6] Roshan Rao, Neil Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S. Song. Evaluating protein transfer learning with tape. In *Advances in Neural Information Processing Systems*, 2019.

[7] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier, 2016.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[9] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional neural networks. In *European Conference on Computer Vision*, pages 818–833, 2014.